Story 1.5: Domain Models - Aggregates & Entities

Status: drafted

Story

As a Developer,
I want Domain models (SignatureRequest aggregate, ValueObjects) implementados,
so that Puedo codificar lógica de negocio pura sin dependencias externas.

Acceptance Criteria

AC1: SignatureRequest Aggregate Root

Given Estructura hexagonal establecida (Story 1.1)
When Creo el aggregate root `SignatureRequest` en `domain/model/aggregate/`
Then

- Clase `SignatureRequest` creada con campos:
  - `id: UUID` (UUIDv7, aggregate root identifier)
  - `customerId: String` (pseudonymized customer ID)
  - `transactionContext: TransactionContext` (Value Object immutable)
  - `status: SignatureStatus` (enum: PENDING, CHALLENGED, SIGNED, ABORTED, EXPIRED)
  - `challenges: List<SignatureChallenge>` (Entity collection, 1-to-many)
  - `routingTimeline: List<RoutingEvent>` (Value Object list, audit trail)
  - `createdAt: Instant`
  - `expiresAt: Instant` (TTL: 15 min default)
  - `signedAt: Instant` (nullable, set on SIGNED)
- Builder pattern implementado para construcción fluida
- Lombok `@Builder`, `@Getter`, `@AllArgsConstructor(access = AccessLevel.PRIVATE)` usados
- No imports de Spring, JPA, Jackson, Kafka (domain purity)

AC2: SignatureChallenge Entity

Given SignatureRequest aggregate definido
When Creo la entity `SignatureChallenge` en `domain/model/entity/`
Then

- Clase `SignatureChallenge` creada con campos:
  - `id: UUID` (UUIDv7, challenge identifier)
  - `channelType: ChannelType` (enum: SMS, PUSH, VOICE, BIOMETRIC)
  - `provider: ProviderType` (enum: TWILIO, ONESIGNAL, VONAGE, BIOMETRIC_SDK)
  - `status: ChallengeStatus` (enum: SENT, PENDING, COMPLETED, FAILED, EXPIRED)
  - `sentAt: Instant` (timestamp when challenge sent)
  - `completedAt: Instant` (nullable, timestamp when completed)
  - `providerProof: ProviderResult` (Value Object, non-repudiation evidence)
  - `errorCode: String` (nullable, provider error code if FAILED)
- Builder pattern implementado
- Método `complete(ProviderResult proof)` → transición PENDING → COMPLETED
- Método `fail(String errorCode)` → transición PENDING → FAILED

AC3: Value Objects (Immutable)

Given Domain models necesitan objetos inmutables
When Creo Value Objects en `domain/model/valueobject/`
Then Existen clases:

- **`TransactionContext`** (Java 21 record):
  - `amount: Money` (Value Object)
  - `merchantId: String`
  - `orderId: String`
  - `description: String`
  - `hash: String` (SHA256 hash, integrity check)
- **`Money`** (Java 21 record):
  - `amount: BigDecimal`
  - `currency: String` (ISO 4217 code: EUR, USD)

    - Método `add(Money other)` → retorna nuevo Money
    - Método `multiply(BigDecimal factor)` → retorna nuevo Money
- **`ProviderResult`** (Java 21 record):
    - `proof: String` (provider response signature/token)
    - `timestamp: Instant`
    - `metadata: Map<String, Object>` (additional provider data)
- **`RoutingEvent`** (Java 21 record):
    - `timestamp: Instant`
    - `eventType: String` (e.g., "FALLBACK_TRIGGERED", "CHALLENGE_SENT")
    - `fromChannel: ChannelType` (nullable)
    - `toChannel: ChannelType` (nullable)
    - `reason: String`

AC4: Enums (Domain Constants)

Given Domain models usan tipos discretos
When Creo enums en `domain/model/valueobject/`
Then Existen enums:

- **`SignatureStatus`**: PENDING, CHALLENGED, SIGNED, ABORTED, EXPIRED
- **`ChallengeStatus`**: SENT, PENDING, COMPLETED, FAILED, EXPIRED
- **`ChannelType`**: SMS, PUSH, VOICE, BIOMETRIC
- **`ProviderType`**: TWILIO, ONESIGNAL, VONAGE, BIOMETRIC_SDK

AC5: SignatureRequest Business Methods

Given SignatureRequest aggregate con estado
When Implemento métodos de negocio en `SignatureRequest`
Then Métodos implementados:

- **`createChallenge(ChannelType channel, ProviderType provider)`**:
    - Valida: solo 1 challenge activo (status PENDING) permitido
    - Crea nuevo `SignatureChallenge` con status SENT
    - Agrega challenge a `this.challenges`
    - Transiciona aggregate status a CHALLENGED
    - Agrega `RoutingEvent` a `routingTimeline`

    - Retorna: `SignatureChallenge` creado
    - Lanza: `DomainException` si ya existe challenge activo
- **`completeSignature(SignatureChallenge challenge)`**:
    - Valida: challenge pertenece a este aggregate
    - Valida: challenge status = COMPLETED
    - Transiciona aggregate status a SIGNED
    - Set `signedAt = Instant.now()`
    - Agrega `RoutingEvent` a `routingTimeline`
    - Retorna: void
    - Lanza: `DomainException` si challenge no COMPLETED o no pertenece
- **`abort(String reason)`**:
    - Transiciona aggregate status a ABORTED
    - Agrega `RoutingEvent` con reason
    - Retorna: void
- **`expire()`**:
    - Valida: `Instant.now().isAfter(expiresAt)`
    - Transiciona aggregate status a EXPIRED
    - Agrega `RoutingEvent` con reason "TTL_EXCEEDED"
    - Retorna: void
    - Lanza: `DomainException` si no expirado todavía

AC6: Domain Exceptions

Given Domain models necesitan excepciones específicas
When Creo domain exceptions en `domain/exception/`
Then Existen clases:

- **`DomainException`** (abstract base class):
    - `message: String`
    - `errorCode: String`
    - Constructor: `DomainException(String message, String errorCode)`
- **`FallbackExhaustedException extends DomainException`**:
    - Constructor: `FallbackExhaustedException(String message)`
    - errorCode: "FALLBACK_EXHAUSTED"

- **`InvalidStateTransitionException extends DomainException`**:
  - Constructor: `InvalidStateTransitionException(String message, SignatureStatus from, SignatureStatus to)`
  - errorCode: "INVALID_STATE_TRANSITION"
- **`ChallengeAlreadyActiveException extends DomainException`**:
  - Constructor: `ChallengeAlreadyActiveException(UUID signatureRequestId)`
  - errorCode: "CHALLENGE_ALREADY_ACTIVE"

AC7: Domain Purity (ArchUnit Validation)

Given Domain models creados
When Ejecuto `HexagonalArchitectureTest.java`
Then

- Test `domainLayerShouldNotDependOnInfrastructure()` pasa
- Test `domainLayerShouldNotDependOnSpring()` pasa
- Test `domainLayerShouldNotDependOnJPA()` pasa
- Test `domainLayerShouldNotDependOnJackson()` pasa
- Test `domainLayerShouldNotDependOnKafka()` pasa
- Ninguna clase en `com.bank.signature.domain` importa:
  - `org.springframework.*`
  - `javax.persistence.*`, `jakarta.persistence.*`
  - `com.fasterxml.jackson.*`
  - `org.apache.kafka.*`

AC8: Unit Tests (Business Logic)

Given SignatureRequest con business methods
When Creo unit tests en `test/java/com/bank/signature/domain/`
Then Tests creados:

- **`SignatureRequestTest.java`**:
  - `testCreateChallenge_Success()` → crea challenge, verifica status CHALLENGED
  - `testCreateChallenge_ThrowsWhenChallengeAlreadyActive()` → lanza exception si challenge PENDING existe
  - `testCompleteSignature_Success()` → completa signature, verifica status SIGNED, signedAt set

    - `testCompleteSignature_ThrowsWhenChallengeNotCompleted()` → lanza exception si challenge no COMPLETED
    - `testAbort_Success()` → aborta signature, verifica status ABORTED
    - `testExpire_Success()` → expira signature si expiresAt pasado, verifica status EXPIRED
    - `testExpire_ThrowsWhenNotExpired()` → lanza exception si TTL no excedido
- **`MoneyTest.java`**:
    - `testAdd_SameCurrency()` → suma correcta
    - `testAdd_DifferentCurrency_ThrowsException()` → lanza exception si currencies diferentes
    - `testMultiply()` → multiplicación correcta
- **`TransactionContextTest.java`**:
    - `testHash_Immutability()` → hash no cambia después de creación (record immutability)

AC9: Builder Pattern Usage Examples

Given SignatureRequest con builder
When Uso builder para crear aggregate
Then Código ejemplo funciona:

```
SignatureRequest request = SignatureRequest.builder()
    .id(UUIDGenerator.generateV7())
    .customerId("pseudonymized-cust-123")
    .transactionContext(new TransactionContext(
        new Money(new BigDecimal("100.00"), "EUR"),
        "merchant-789",
        "order-456",
        "Payment for Order #456",
        "sha256-hash-xyz"
    ))
    .status(SignatureStatus.PENDING)
    .challenges(new ArrayList<>())
    .routingTimeline(new ArrayList<>())
    .createdAt(Instant.now())
    .expiresAt(Instant.now().plus(Duration.ofMinutes(15)))
    .build();
```

AC10: Lombok Configuration

Given Domain models usan Lombok
When Configuro `lombok.config` en project root
Then Archivo `lombok.config` creado con:

```
lombok.addLombokGeneratedAnnotation = true
lombok.anyConstructor.addConstructorProperties = false
lombok.fieldDefaults.defaultPrivate = true
lombok.fieldDefaults.defaultFinal = true
```

- `@Generated` annotation agregada por Lombok (excluye de code coverage)
- Builder pattern default private/final para immutability

AC11: Package Structure

Given Domain models creados
When Reviso estructura de packages
Then Estructura es:

```
src/main/java/com/bank/signature/domain/
├── model/
│   ├── aggregate/
│   │   └── SignatureRequest.java
│   ├── entity/
│   │   └── SignatureChallenge.java
│   └── valueobject/
│       ├── TransactionContext.java (record)
│       ├── Money.java (record)
│       ├── ProviderResult.java (record)
│       ├── RoutingEvent.java (record)
│       ├── SignatureStatus.java (enum)
│       ├── ChallengeStatus.java (enum)
│       ├── ChannelType.java (enum)
│       └── ProviderType.java (enum)
├── exception/
│   ├── DomainException.java
│   ├── FallbackExhaustedException.java
│   ├── InvalidStateTransitionException.java
│   └── ChallengeAlreadyActiveException.java
└── service/
    └── (placeholder para domain services futuros)
```

AC12: Documentation & Testing Summary

Given Story 1.5 implementado
When Actualizo documentación
Then

- **README.md** actualizado con sección "Domain Models" (package structure, examples)
- **CHANGELOG.md** actualizado con Story 1.5 entry
- Unit tests en `src/test/java/com/bank/signature/domain/` (no Spring, pure JUnit 5)
- Test coverage > 80% para domain models (medido con JaCoCo)
- JavaDoc en métodos públicos de `SignatureRequest`, `SignatureChallenge`

Tasks / Subtasks

Task 1: Create Enums (Domain Constants) (AC: #4)

- ◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/SignatureStatus.java`
  - ◯ Define enum values: PENDING, CHALLENGED, SIGNED, ABORTED, EXPIRED
  - ◯ Add JavaDoc describing each status
- ◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/ChallengeStatus.java`
  - ◯ Define enum values: SENT, PENDING, COMPLETED, FAILED, EXPIRED
  - ◯ Add JavaDoc describing each status
- ◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/ChannelType.java`
  - ◯ Define enum values: SMS, PUSH, VOICE, BIOMETRIC
  - ◯ Add JavaDoc describing each channel
- ◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/ProviderType.java`
  - ◯ Define enum values: TWILIO, ONESIGNAL, VONAGE, BIOMETRIC_SDK
  - ◯ Add JavaDoc describing each provider

Task 2: Create Value Objects (Immutable Records) (AC: #3)

- ◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/Money.java`
  - ◯ Define Java 21 record with fields: amount (BigDecimal), currency (String)
  - ◯ Implement `add(Money other)` method with currency validation
  - ◯ Implement `multiply(BigDecimal factor)` method
  - ◯ Add validation in compact constructor (non-null, currency not empty, amount >= 0)
  - ◯ Add JavaDoc with usage examples
- ◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/TransactionContext.java`
  - ◯ Define Java 21 record with fields: amount (Money), merchantId, orderId, description, hash
  - ◯ Add validation in compact constructor (non-null fields, hash SHA256 format)
  - ◯ Add JavaDoc
- ◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/ProviderResult.java`
  - ◯ Define Java 21 record with fields: proof (String), timestamp (Instant), metadata (Map<String, Object>)
  - ◯ Add validation in compact constructor (non-null proof, timestamp)
  - ◯ Add JavaDoc
- ◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/RoutingEvent.java`
  - ◯ Define Java 21 record with fields: timestamp, eventType, fromChannel, toChannel, reason
  - ◯ Add validation in compact constructor (non-null timestamp, eventType)
  - ◯ Add JavaDoc

Task 3: Create Domain Exceptions (AC: #6)

- ◯ Create `src/main/java/com/bank/signature/domain/exception/DomainException.java`
  - ◯ Define abstract base class extending RuntimeException
  - ◯ Add fields: errorCode (String)

- ◯ Add constructor: DomainException(String message, String errorCode)
- ◯ Add getters for errorCode

◯ Create `src/main/java/com/bank/signature/domain/exception/FallbackExhaustedException.java`

- ◯ Extend DomainException
- ◯ Constructor with message, hardcode errorCode "FALLBACK_EXHAUSTED"

◯ Create `src/main/java/com/bank/signature/domain/exception/InvalidStateTransitionException.java`

- ◯ Extend DomainException
- ◯ Constructor with message, SignatureStatus from, SignatureStatus to
- ◯ errorCode: "INVALID_STATE_TRANSITION"

◯ Create `src/main/java/com/bank/signature/domain/exception/ChallengeAlreadyActiveException.java`

- ◯ Extend DomainException
- ◯ Constructor with UUID signatureRequestId
- ◯ errorCode: "CHALLENGE_ALREADY_ACTIVE"
- ◯ Message format: "Signature request {id} already has an active challenge"

Task 4: Create SignatureChallenge Entity (AC: #2)

◯ Create `src/main/java/com/bank/signature/domain/model/entity/SignatureChallenge.java`

- ◯ Add Lombok annotations: @Builder, @Getter, @AllArgsConstructor(access = AccessLevel.PRIVATE)
- ◯ Define fields: id, channelType, provider, status, sentAt, completedAt, providerProof, errorCode
- ◯ Implement `complete(ProviderResult proof)` method
  - ◯ Validate status is PENDING (throw InvalidStateTransitionException if not)
  - ◯ Set status = COMPLETED
  - ◯ Set completedAt = Instant.now()
  - ◯ Set providerProof = proof
- ◯ Implement `fail(String errorCode)` method

- ◯ Validate status is PENDING
- ◯ Set status = FAILED
- ◯ Set this.errorCode = errorCode

◯ Add JavaDoc for public methods

Task 5: Create SignatureRequest Aggregate Root (AC: #1, #5)

◯ Create `src/main/java/com/bank/signature/domain/model/aggregate/SignatureRequest.java`

- ◯ Add Lombok annotations: @Builder, @Getter, @AllArgsConstructor(access = AccessLevel.PRIVATE)
- ◯ Define fields: id, customerId, transactionContext, status, challenges, routingTimeline, createdAt, expiresAt, signedAt
- ◯ Implement `createChallenge(ChannelType channel, ProviderType provider)` method
  - ◯ Validate: no challenge with status PENDING exists (throw ChallengeAlreadyActiveException if exists)
  - ◯ Create new SignatureChallenge with status SENT
  - ◯ Add challenge to this.challenges list
  - ◯ Set this.status = CHALLENGED
  - ◯ Add RoutingEvent to routingTimeline (eventType: "CHALLENGE_SENT")
  - ◯ Return created SignatureChallenge
- ◯ Implement `completeSignature(SignatureChallenge challenge)` method
  - ◯ Validate: challenge exists in this.challenges (throw DomainException if not)
  - ◯ Validate: challenge.status == COMPLETED (throw InvalidStateTransitionException if not)
  - ◯ Set this.status = SIGNED
  - ◯ Set this.signedAt = Instant.now()
  - ◯ Add RoutingEvent to routingTimeline (eventType: "SIGNATURE_COMPLETED")
- ◯ Implement `abort(String reason)` method
  - ◯ Set this.status = ABORTED
  - ◯ Add RoutingEvent to routingTimeline (eventType: "SIGNATURE_ABORTED", reason: reason)
- ◯ Implement `expire()` method

- ◯ Validate: Instant.now().isAfter(expiresAt) (throw DomainException if not expired)
- ◯ Set this.status = EXPIRED
- ◯ Add RoutingEvent to routingTimeline (eventType: "SIGNATURE_EXPIRED", reason: "TTL_EXCEEDED")

◯ Add JavaDoc for all public methods with examples

Task 6: Create UUIDv7 Generator Utility (Supporting Class)

◯ Create `src/main/java/com/bank/signature/domain/model/valueobject/UUIDGenerator.java`

- ◯ Implement `generateV7()` static method
- ◯ UUIDv7 format: 48-bit timestamp + 4-bit version (0111) + 74-bit random
- ◯ Add JavaDoc explaining UUIDv7 benefits (time-sortable, better B-tree performance)
- ◯ Add unit test: `UUIDGeneratorTest.testGenerateV7_IsSortable()`

Task 7: Configure Lombok (AC: #10)

◯ Create `lombok.config` in project root

- ◯ Add config: `lombok.addLombokGeneratedAnnotation = true`
- ◯ Add config: `lombok.anyConstructor.addConstructorProperties = false`
- ◯ Add config: `lombok.fieldDefaults.defaultPrivate = true`
- ◯ Add config: `lombok.fieldDefaults.defaultFinal = true`

Task 8: Update ArchUnit Tests (AC: #7)

◯ Update `src/test/java/com/bank/signature/HexagonalArchitectureTest.java`

- ◯ Add test: `domainLayerShouldNotDependOnSpring()`
  - ◯ Rule: classes in "..domain.." should not depend on "..springframework.."
- ◯ Add test: `domainLayerShouldNotDependOnJPA()`
  - ◯ Rule: classes in "..domain.." should not depend on "..jakarta.persistence.." or "..javax.persistence.."
- ◯ Add test: `domainLayerShouldNotDependOnJackson()`
  - ◯ Rule: classes in "..domain.." should not depend on "..fasterxml.jackson.."
- ◯ Add test: `domainLayerShouldNotDependOnKafka()`
  - ◯ Rule: classes in "..domain.." should not depend on "..apache.kafka.."

- ◯ Run all tests to verify domain purity

Task 9: Create Unit Tests for Domain Models (AC: #8)

- ◯ Create `src/test/java/com/bank/signature/domain/model/aggregate/SignatureRequestTest.java`
  - ◯ Test: `testCreateChallenge_Success()` → verify status CHALLENGED, challenge added to list, routingTimeline updated
  - ◯ Test: `testCreateChallenge_ThrowsWhenChallengeAlreadyActive()` → verify ChallengeAlreadyActiveException thrown
  - ◯ Test: `testCompleteSignature_Success()` → verify status SIGNED, signedAt set, routingTimeline updated
  - ◯ Test: `testCompleteSignature_ThrowsWhenChallengeNotCompleted()` → verify InvalidStateTransitionException thrown
  - ◯ Test: `testCompleteSignature_ThrowsWhenChallengeNotBelongsToAggregate()` → verify DomainException thrown
  - ◯ Test: `testAbort_Success()` → verify status ABORTED, routingTimeline updated with reason
  - ◯ Test: `testExpire_Success()` → verify status EXPIRED when TTL exceeded
  - ◯ Test: `testExpire_ThrowsWhenNotExpired()` → verify DomainException thrown if TTL not exceeded
- ◯ Create `src/test/java/com/bank/signature/domain/model/valueobject/MoneyTest.java`
  - ◯ Test: `testAdd_SameCurrency()` → verify correct sum
  - ◯ Test: `testAdd_DifferentCurrency_ThrowsException()` → verify exception thrown
  - ◯ Test: `testMultiply()` → verify correct multiplication
  - ◯ Test: `testConstructor_NegativeAmount_ThrowsException()` → verify validation
- ◯ Create `src/test/java/com/bank/signature/domain/model/valueobject/TransactionContextTest.java`
  - ◯ Test: `testImmutability()` → verify fields cannot be modified (record immutability)
  - ◯ Test: `testHash_NotNull()` → verify hash validation in constructor
- ◯ Create

`src/test/java/com/bank/signature/domain/model/entity/SignatureChallengeTest.java`

- ◯ Test: `testComplete_Success()` → verify status COMPLETED, completedAt set, providerProof set
- ◯ Test: `testComplete_ThrowsWhenNotPending()` → verify InvalidStateTransitionException
- ◯ Test: `testFail_Success()` → verify status FAILED, errorCode set

◯ Run all tests with `mvn test` to verify > 80% coverage (JaCoCo)

Task 10: Update Documentation (AC: #12)

◯ Update `README.md`

- ◯ Add "Domain Models" section after "Vault Secret Management"
- ◯ Include package structure diagram
- ◯ Include builder pattern usage example
- ◯ Link to architecture docs (02-hexagonal-structure.md)

◯ Update `CHANGELOG.md`

- ◯ Add Story 1.5 entry under [Unreleased]
- ◯ List added features: SignatureRequest aggregate, SignatureChallenge entity, 4 Value Objects (Money, TransactionContext, ProviderResult, RoutingEvent), 4 enums, 4 domain exceptions
- ◯ List technical details: Java 21 records, Lombok @Builder, domain purity (ArchUnit validated), unit tests (80%+ coverage)

◯ Update `docs/architecture/02-hexagonal-structure.md`

- ◯ Add concrete examples for SignatureRequest, SignatureChallenge in "Domain Layer" section
- ◯ Add example business logic methods (createChallenge, completeSignature)

Implementation Highlights

Domain-Driven Design (DDD) Patterns

1. **Aggregate Root**: `SignatureRequest` with identity, encapsulated entities (`challenges`), and business invariants
2. **Entity**: `SignatureChallenge` with identity and lifecycle
3. **Value Objects**: Immutable records (`Money`, `TransactionContext`, `ProviderResult`, `RoutingEvent`)

4. **Domain Exceptions**: Business-specific exceptions (`FallbackExhaustedException`, `InvalidStateTransitionException`)
5. **Ubiquitous Language**: Enums match business terminology (`ChannelType`, `ProviderType`)

Java 21 Features

- **Records**: Immutable Value Objects with compact syntax
- **Pattern Matching**: (Planned for future use in domain services)
- **Sealed Classes**: (Planned for future hierarchies)

Lombok Configuration

- **@Builder**: Fluent API for aggregate construction
- **@Getter**: Immutable read-only access
- **AccessLevel.PRIVATE**: Constructor only via Builder (enforces invariants)

ArchUnit Validation

- **Domain Purity**: No dependencies on Spring, JPA, Jackson, Kafka
- **Hexagonal Boundaries**: Domain cannot depend on infrastructure

Source Tree (Files to Create/Modify)

Files to Create

Domain Models (11 files):

- `src/main/java/com/bank/signature/domain/model/aggregate/SignatureRequest.java`
- `src/main/java/com/bank/signature/domain/model/entity/SignatureChallenge.java`
- `src/main/java/com/bank/signature/domain/model/valueobject/Money.java`
- `src/main/java/com/bank/signature/domain/model/valueobject/TransactionContext.java`
- `src/main/java/com/bank/signature/domain/model/valueobject/ProviderResult.java`
- `src/main/java/com/bank/signature/domain/model/valueobject/RoutingEvent.java`
- `src/main/java/com/bank/signature/domain/model/valueobject/UUIDGenerator.j`

ava

- src/main/java/com/bank/signature/domain/model/valueobject/SignatureStatus.java
- src/main/java/com/bank/signature/domain/model/valueobject/ChallengeStatus.java
- src/main/java/com/bank/signature/domain/model/valueobject/ChannelType.java
- src/main/java/com/bank/signature/domain/model/valueobject/ProviderType.java

Domain Exceptions (4 files):

- src/main/java/com/bank/signature/domain/exception/DomainException.java
- src/main/java/com/bank/signature/domain/exception/FallbackExhaustedException.java
- src/main/java/com/bank/signature/domain/exception/InvalidStateTransitionException.java
- src/main/java/com/bank/signature/domain/exception/ChallengeAlreadyActiveException.java

Unit Tests (5 files):

- src/test/java/com/bank/signature/domain/model/aggregate/SignatureRequestTest.java
- src/test/java/com/bank/signature/domain/model/entity/SignatureChallengeTest.java
- src/test/java/com/bank/signature/domain/model/valueobject/MoneyTest.java
- src/test/java/com/bank/signature/domain/model/valueobject/TransactionContextTest.java
- src/test/java/com/bank/signature/domain/model/valueobject/UUIDGeneratorTest.java

Configuration (1 file):

- lombok.config

Files to Modify

- `src/test/java/com/bank/signature/HexagonalArchitectureTest.java` - Add domain purity tests
- `README.md` - Add "Domain Models" section
- `CHANGELOG.md` - Add Story 1.5 entry
- `docs/architecture/02-hexagonal-structure.md` - Add concrete examples

References to Existing Documentation

- **Architecture**: `docs/architecture/02-hexagonal-structure.md` (Domain Layer package structure)
- **Database Schema**: `docs/architecture/03-database-schema.md` (SignatureRequest, SignatureChallenge table definitions)
- **Tech Spec Epic 1**: `docs/sprint-artifacts/tech-spec-epic-1.md` (Technology stack, DDD patterns)

Testing Strategy

Unit Tests (Pure JUnit 5, No Spring)

1. **Aggregate Tests**: SignatureRequestTest (8 test methods)
2. **Entity Tests**: SignatureChallengeTest (4 test methods)
3. **Value Object Tests**: MoneyTest (4 test methods), TransactionContextTest (2 test methods)
4. **ArchUnit Tests**: Domain purity validation (5 architecture rules)

Target Coverage: > 80% line coverage (JaCoCo)

Test Execution:

```
mvn test
mvn jacoco:report
# View coverage: target/site/jacoco/index.html
```

Definition of Done

- ◯ All 12 Acceptance Criteria verified
- ◯ SignatureRequest aggregate implemented with 4 business methods
- ◯ SignatureChallenge entity implemented with 2 methods (complete, fail)
- ◯ 4 Value Objects implemented as Java 21 records (Money, TransactionContext,

ProviderResult, RoutingEvent)

- ◯ 4 Enums implemented (SignatureStatus, ChallengeStatus, ChannelType, ProviderType)
- ◯ 4 Domain Exceptions implemented
- ◯ UUIDv7 generator utility implemented
- ◯ lombok.config created with recommended settings
- ◯ ArchUnit tests updated and passing (domain purity validated)
- ◯ Unit tests created (21+ test methods) with > 80% coverage
- ◯ README.md updated with "Domain Models" section
- ◯ CHANGELOG.md updated with Story 1.5 entry
- ◯ docs/architecture/02-hexagonal-structure.md updated with examples
- ◯ No domain classes import Spring/JPA/Jackson/Kafka (verified by ArchUnit)
- ◯ mvn test passes without errors
- ◯ Code review approved

Dev Agent Record

Context Reference

Agent Model Used

Claude Sonnet 4.5

Debug Log References

Completion Notes List

File List

Created:

Modified:

Deleted:

Change Log

Date	Author	Change
2025-11-26	BMAD SM Agent	Story 1.5 draft created: Domain Models - Aggregates & Entities (DDD patterns)